

Fast algorithm finding the shortest reset words

Andrzej Kisielewicz^{1,2}, Jakub Kowalski¹, and Marek Szykuła¹

¹ Department of Mathematics and Computer Science, University of Wrocław, Poland

² Institute of Mathematics and Computer Science, University of Opole, Poland
 andrzej.kisielewicz@math.uni.wroc.pl, {kot,msz}@ii.uni.wroc.pl

Abstract. In this paper we present a new fast algorithm finding minimal reset words for finite synchronizing automata. The problem is known to be computationally hard, and our algorithm is exponential. Yet, it is faster than the algorithms used so far and it works well in practice. The main idea is to use a bidirectional BFS and radix (Patricia) tries to store and compare resulted subsets. We give both theoretical and practical arguments showing that the branching factor is reduced efficiently. As a practical test we perform an experimental study of the length of the shortest reset word for random automata with n states and 2 input letters. We follow Skvorsov and Tipikin, who have performed such a study using a SAT solver and considering automata up to $n = 100$ states. With our algorithm we are able to consider much larger sample of automata with up to $n = 300$ states. In particular, we obtain a new more precise estimation of the expected length of the shortest reset word $\approx 2.5\sqrt{n-5}$.

Keywords: Synchronizing DFA, synchronizing word, Černý conjecture, radix trie

1 Introduction

We deal with deterministic finite automata $A = \langle Q, \Sigma, \delta \rangle$ with the state set Q , the input alphabet Σ , and the transition function $\delta : Q \times \Sigma \rightarrow Q$. The action of Σ on Q given by δ is denoted simply by concatenation: $\delta(q, a) = qa$. This action extends naturally to the action qw of the words for any $w \in \Sigma^*$. If $|Qw| = 1$, that is, the image of Q by w consists of a single state, then w is called a *reset* (or *synchronizing*) word for A , and A itself is called *synchronizing*.

The Černý conjecture states that every synchronizing automaton A with n states has a reset word of length $\leq (n-1)^2$. This conjecture was formulated by Černý in 1964, and is considered the most longstanding open problem in the combinatorial theory of finite automata. So far, the conjecture has been proved only for a few special classes of automata and a general cubic upper bound has been established (see Volkov [22] for general motivation and an excellent survey of the results, and Trahtman [19] for a recently found new cubic bound). Using computers the conjecture has been verified for small automata with 2 letters and $n \leq 10$ states (and with $q \leq 4$ letters and $n \leq 7$ states [21]; see also [1] for $n = 9$ states). It is known that, in general, the problem is computationally hard, since it involves an NP-hard decision problem. Recently, it has been shown that the problem of finding the length of the shortest reset word is $\text{FNP}^{\text{NP}[\log]}$ -complete, and the related decision problem is both NP- and coNP-hard [12] (cf. also [2] and [9,10]).

On the other hand, there are several theoretical and experimental results showing that most synchronizing automata have relatively short reset words and those slowly synchronizing (with the shortest reset words of quadratic length) are rather exceptional [1]. An old result of Higgins [7] on products in transformation semigroups shows that a random automaton with an alphabet of size larger than $2n$ has, with high probability, a reset word of length $\leq 2n$. More recently, it was proved that a random automaton with n states over an alphabet with $n^{0.5+\epsilon}$, with high probability, is synchronizing and satisfies the Černý conjecture [18]. In computing reset words,

either exponential algorithms finding the shortest reset words [17,21,8] or polynomial heuristics finding relatively short reset words [6,14,15,16,21] are widely used. The standard approach is the breadth-first-search method which starts from the set of all states of the given automaton and forms images applying letter transformations until a singleton is reached. Another approach is the semigroup algorithm which generates all transitions for increasing words lengths until the synchronizing one is found [21]. Based on these ideas computation packages have been created (TESTAS [20] and recently developed COMPAS [3]). In [15], Roman uses a genetic algorithm to find a reset word of randomly generated automata and thus obtains upper bounds on the length of the shortest reset word.

A new interesting approach for finding the exact length using a SAT-solver has been applied recently by Skvorsov and Tipikin [17]. The problem of determining if an automaton has a reset word of length at most l is reduced to the SAT problem and the binary search is performed. Using this approach, the following experimental study is done. For chosen numbers n of states from the interval $[1, 100]$ random automata with 2 input letters are generated, checked if they are synchronizing, and if so, the shortest reset word is computed. The results directly contradict the conjecture made by Roman [15] that the mean length of the shortest reset word for a random n -state synchronizing automaton is linear and almost equal to $0.486n$. Skvorsov and Tipikin argue that their experiment based on a larger set of data shows that this length is actually sublinear and $\approx 1.95n^{0.55}$. They have generated randomly and check 2000 automata for each $n \in \{1, 2, \dots, 20, 25, 30, \dots, 50\}$, 500 automata for each $n \in \{55, 60, 65, 70\}$, and 200 automata for each $n \in \{75, 80, \dots, 100\}$.

In this paper we present a new algorithm based on a bidirectional breadth-first-search. Implementing this idea requires to solve efficiently the problem of storing and comparing resulted subsets of states. To this aim radix tries (also known as Patricia tries [11]) are used. We analyse the algorithm from both theoretical and practical sides. As the first test of efficiency we have performed experiments analogous to those done by Skvorsov and Tipikin. We were able to generate and check one million automata for each $n \leq 100$, 10000 automata for each further $n \leq 220$, and 1000 automata for each n in the set $\{225, 230, \dots, 300\}$. Our data confirm the hypothesis that the expected length of the shortest reset word is sublinear, but show that more precise is a smaller approximation $\approx 2.5\sqrt{n-5}$. In addition, the larger set of data enables us to estimate the error and to show that for our approximation with high probability the error is very small. We also verify and discuss other results and claims of [17].

Our algorithm is suitable to find the shortest reset words, not only their lengths. Curiously, it works in polynomial time for known slowly synchronizing automata series [1]. Since there are opinions that random automata with more than 2 input letters may exhibit a different behavior, we continue our experiment admitting various alphabets of size $k \geq 2$. The results will be presented in the forthcoming extended version of the paper.

2 Algorithm

The algorithm gets an automaton $A = \langle Q, \Sigma, \delta \rangle$ with n states and k input letters. First, A is checked if it is synchronizing using the well known (and efficient) algorithm [5]. To find the shortest synchronizing word, the main idea is to perform a bidirectional breadth-first-search. The steps of the standard BFS and inverse BFS (IBFS) are performed alternately. The BFS starts from the set of all states Q and computes a list of possible images. The IBFS starts from all the single states and computes the list of possible preimages. The branching factor of both of them is k . In each step, both BFS and IBFS, computing images or preimages, respectively, creates a list of *subsets* of Q , which are called BFS-list and IBFS-list, respectively. We make use of the following

Fact 1. *Let S, T be subsets of Q . If $S \subseteq T$, and w_s and w_t are the shortest words over Σ such that $|Sw_s| = 1$ and $|Tw_t| = 1$, then w_s is not longer than w_t .*

Thus, after each step of BFS we can reduce the BFS-list to contain only minimal subsets. Each step of BFS consists of the following sub-steps. For every subset S in BFS-list we form the images of S by all the letters, and for every such image T we check whether we have already visited any subset of T . To this aim we maintain the list of subsets visited by BFS, called BFS-visited list. If a subset of T is in the BFS-visited list, then T is removed; otherwise it is removed and inserted into the BFS-visited list and into the new, created during this step, BFS-list (corresponding to the present level of BFS). This is repeated until the former BFS-list is empty. We note that the new resulting BFS list is partially ordered in the sense that each inserted subset never contains any earlier inserted subset (otherwise it is removed on an earlier step). This helps to reduce the BFS-list by removing subsets that contain other subsets in the list in an efficient way (which will be described later).

Each step of IBFS is performed in an analogous (dual) way and results with the IBFS-list reduced to contain only maximal subsets. After each step of BFS or IBFS the current lists are compared whether there is a subset in the IBFS-list containing a subset in the BFS-list. If so, the algorithm terminates. The number of done steps is the length of the shortest reset word. If we store also the sequences of letters used to form images, we may output also the reset word itself.

2.1 Radix tries

In order to check the required conditions in a fast way, we need a data structure for storing subsets of Q and checking for a given subset S if it contains any subset inserted to the structure (subset checking). We use the well known radix tries (Patricia tries) to this aim [11]. Here a *radix trie* is a binary tree of the maximal depth n which stores subsets of a given n -set Q in its leaves. Having a fixed order of elements $q_1, \dots, q_n \in Q$, each subset S of Q encodes a path from the root to a leaf in the natural way: after i steps the path goes to the right child whenever $q_i \in S$, and goes to the left, otherwise. A radix trie is *compressed* in the sense that instead it stores a subset in the first node that determines uniquely the subset in the stored collection (no other subset shares the same path as a prefix).

The insert operation is natural and can be performed in at most n steps creating at most n nodes. The subset checking operation is performed by a depth-first-search checking if the given subset S contains a subset stored in the visited leaf. The search does not need to branch into the right child of a node if the checked subset S does not contain the corresponding state. It performs nm operations in the worst case, where m is the number of subsets stored in a trie, but in practice the performance is much better (we discuss it later). The superset operation (for IBFS) is done in the dual way.

In our algorithm we use radix tries for three different tasks. First of all we use them to maintain BFS- and IBFS-visited lists and to perform, respectively, subsets or superset checking. Next, we use radix tries to reduce current BFS- and IBFS-lists. These are standard lists, but to reduce them we form an empty auxiliary radix trie and insert the elements of, say, BFS-list in the backward order. Before inserting each element a DFS-check for subsets is performed. Due to the fact (mentioned earlier) that the list is partially ordered (larger sets precede their subsets on the list) this leads to the minimal list, that is, no subset contains any other subset. The IBFS-list is reduced in the dual way. The only substantial difference is that we remove the auxiliary IBFS-trie after using, while the auxiliary BFS-trie is kept to be used for the next task. Namely, in checking if the current IBFS-list has an element containing a subset in the BFS-list, we iterate over the

elements of the former and perform subset checking operation using the auxiliary BFS radix trie (which contains at the moment the same elements as the current BFS-list).

2.2 Visited subsets

The tries of visited subsets are the largest structures in the algorithm, so we try to keep them minimal. When we insert a subset which is contained in an already stored subset, we can remove the larger one. If we reach a node with the larger set we can just replace it by the smaller one and the size of a trie does not grow. This is the case when the prefixes of comparable subsets are equal. We do not look for and do not remove other larger sets during the insert operation, since we have found it is too time-expensive. Yet, we sometimes reduce the sizes of the BFS- and IBFS-visited tries by rebuilding them in the backward order to obtain minimal collections of subsets.

If these tries grow too large such that we run out of memory, then we use a hybrid DFS and BFS method. This procedure starts from the list of subsets obtained by the last BFS step. If the list is short then it performs one BFS step as in the previous part, but no visited subsets are remembered nor checked anymore. If the list is long, then it is split into a few smaller lists. Then the procedure calls recursively for each of the lists starting from one containing the smallest subsets. The maximal depth of the searching is the currently smallest length of a found synchronizing word. In practice, this stage is used only for very large automata.

2.3 Heuristics

We use also some heuristics to optimize the algorithm. Before each step we sort the BFS-list in the ascending set sizes order and the IBFS-list in the descending order. This helps in skipping redundant subsets, since it implies that the smaller subsets are checked before the larger and the resulting list (before minimalization) is smaller. Also, using this, we insert less subsets into the tries containing visited subsets.

To decide which step should be performed, the BFS or the IBFS, we compare the sizes of both current lists and perform the step for the smaller one. We find it is faster to do with a slight overhead of the BFS. We use $k = |\Sigma|$ times larger weight for the IBFS-list. The BFS step is slightly faster since in random automata overall size of subsets after x steps are smaller than the overall n minus size after x steps if the IBFS.

Random automata are usually not strongly connected, but contain a single strongly connected component, and have the property that, when performing the BFS the nodes outside the component become quickly unreachable. So at the beginning of the algorithm we perform a few steps of the standard BFS and construct a reduced automaton without these unreachable states. Then the algorithm is called for the reduced automaton. The reduction usually gets rid of about 20% states for random automata and after a few steps of BFS a strongly connected one is usually obtained (we return to this issue in the next section).

2.4 Expected time of subsets checking

The most time-consuming operation in our algorithm is subset checking in radix tries. We have tried to estimate theoretically an expected time of this operation. Suppose that T is a radix trie containing m random subsets of an n -element universe Q . We assume that a random subset S of Q contains each element of Q with probability p (this is referred to as the Bernoulli model). Using this model we have proved the following.

Theorem 1 *For a random subset S in the Bernoulli model the expected searching time for a subset of S in the trie T is $O(m^{\log_2(1+p)}/p)$.*

Observe, that this results shows an important improvement over the trivial $O(m)$ comparisons. For example in the uniform model ($p = 0.5$), we have $O(m^{\log_2 3/2}) \subseteq O(m^{0.585})$. We also note that the expected height of a trie with m subsets inserted is known to be $O(\log_2 m)$ (see [4]). The expected time of the insert operation is much faster since it depends linearly on the height of a trie.

In the practical execution of the algorithm the probabilities of subset occurrences are however quite different than in the Bernoulli model. The following are few significant differences (established experimentally):

1. When a subset of S is found in the trie of visited subsets searching is terminated and this situation comes more frequently when executing the algorithm than in the Bernoulli model.
2. The numbers of subsets stored in tries are smaller than the number of subsets inserted into them. This is because of the procedure of replacing subsets in case of detected inclusion.
3. The probability distribution in the real performance is not constant. The sizes of subsets reached by the BFS decrease rapidly in the first few steps. This can be compared with decreasing values of p in the Bernoulli model.

These differences are in favor of the real performance and makes the algorithm to work faster.

2.5 Expected running time

Bounding the expected running time of the algorithm is difficult due to many optimizations. We can however compute a bound for the bidirectional BFS with radix tries and compare it with the standard BFS. We use an assumption that the Bernoulli model considered earlier with small p is not better than the real distribution model.

Then we are able to get the estimation of the expected running time of the form $O(\ln^2 k^{2l/3})$. The experiments performed showed that the real expected running time is much smaller. The reductions of lists to incomparable sets and subset checking work well, and the effective branching factor is considerably less than k .

We have also checked the performance of the algorithm for known *slowly synchronizing automata*: the Černý automata and the automata $\mathcal{W}_n, \mathcal{D}'_n, \mathcal{D}''_n$ and \mathcal{B}_n considered in [1]. Surprisingly, all of these are quickly taken by the IBFS. This is due to the checking for visited subsets and the fact that IBFS-list contains only one subset during all the steps except of the first. The number of stored visited subsets after l steps is so $O(l + n)$ and a subset-checking does not exceed $O(n(n + l))$ operations in the worst case. Thus the running time for them is polynomial and can be roughly estimated as $O(\ln(n + l)) = O(n^5)$.

3 Experiments

We performed a series of the following experiments for various $n \leq 300$. For a given n , we generate a random automaton A with n states and 2 input letters, check whether A is synchronizing and if so, we find the minimal length of a reset word using the algorithm described in Section 2. On the basis of the obtained results we estimate the expected length of the shortest reset word.

3.1 Computations

In the experiments we have used the standard model of random automata, where for each state and each letter all the possible transitions are equiprobable. A random automaton with n states and 2 input letters can be then represented as a sequence of $2n$ uniformly random natural numbers from $[0, n - 1]$. To generate high quality random sequences we have used the WELL number generator [13] (variants 1024 and 19937) seeded by random bytes from $\hat{\text{dev}}/\text{random}$ device. We have computed exact results for automata up to 7 states by checking all of them. For each $8 \leq n \leq 100$ we checked one million automata, for each $101 \leq n \leq 220$ we checked 10000 automata and 1000 for each n in $\{225, 230, \dots, 300\}$.

The computations have been performed mostly on 16 computers with Intel(R) Core(TM) i7-2600 CPU 3.40GHz 4 cores and 1GB of RAM with occasionally support of up to 15 computers with Intel(R) Core(TM) i3 CPU 540 3.07GHz 2 cores and 4GB of RAM. The algorithm was implemented in C++ and compiled with g++. Distributed computations were managed by a dedicated server and clients applications written in Python.

3.2 Efficiency.

The average computation time is about 100 or 1000 times faster than the time of Trahtman's program TESTAS [20,21] for automata with 50 states. The reduction to SAT used in [17] seemed to be the fastest recently known algorithm and the given average time for 50 states automata is 2.7 seconds, and for 100 states automata is 70 seconds. Our comparable results are less then 0.006 and 0.07 seconds, respectively (we have used faster machines but the resulting speedup should be not more than a few times). The table 1 presents the comparison of the average computation time and the maximum computation time for random automata from the experimental data set.

Table 1. The comparison of average computation time and the maximum time for random automata.

n	50	100	150	200	250	300
TESTAS ([20])	1.4 s	time-out	–	–	–	–
SAT reduction ([17])	2.7 s	70 s	–	–	–	–
Our average time	0.005 s	0.065 s	2.65 s	54.4 s	12 min 18 s	1 h 2 min
Our maximum time	0.37 s	33.78 s	360 s	1 h 9 min	3 h 41 min	18 h 19 min

The average times are relatively small because of rare occurrences of „hardly” synchronizable automata. So we present also the maximum computation time which is greatly longer than the average since it is much dependent of an automaton. Our experiment did not find any really „hard” example.

3.3 General results

Our experiment confirms that for the standard random automata model $\mathbb{A}(n)$ on the binary alphabet the probability that the automaton is synchronizing seems to tend to 1 as the number n of states grows:

$$P(\mathbb{A}(n) \text{ is synchronizing}) \xrightarrow{n \rightarrow \infty} 1.$$

This conjecture is posed in [17], but we have heard it earlier from Peter Cameron during BCC conference in Exeter 2011. For $n = 100$, 2250 of one million automata turned out to be non-synchronizing (0.225%), and for $n = 300$, only one of 1000 automata. Based on our experiments,

the line of synchronization chance in Figure 1 for automata with less then 100 states forms a smooth curve which is very likely to converge to 1.

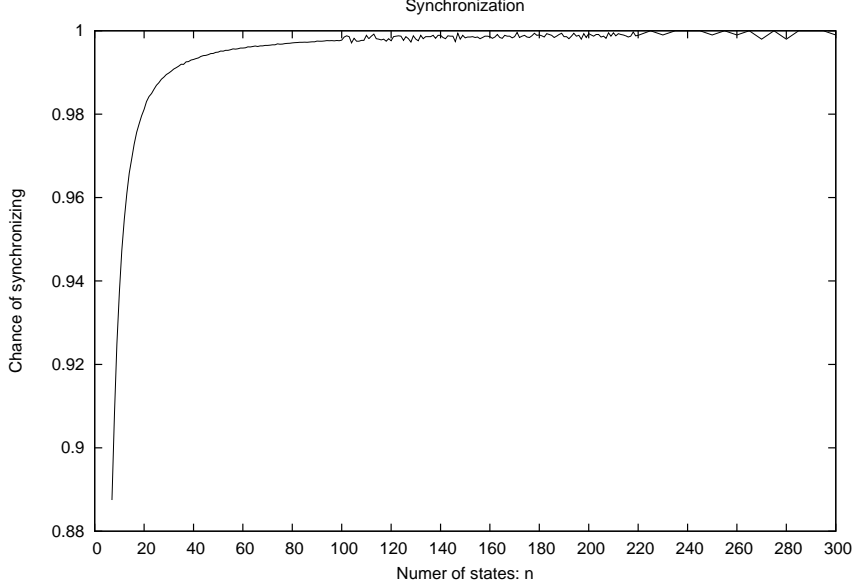


Fig. 1. Experimental values of synchronization probability.

We observe also that random automata are mostly not strongly connected. Moreover, if an automaton is synchronizing then the expected size of the strongly connected sink component seems to tend to the value $\approx 0.7987n$. We also noted that the average length of the minimal synchronizing word in a random automaton is usually a little larger than the length in the strongly connected automaton formed by its sink component.

3.4 The expected length of the shortest reset word

The main result of the experiments is the estimation of the expected minimal length of a synchronizing word. We consider the sequence of random variables $\ell(n)$ defined as the length of the shortest reset word for a synchronizing automata with n states. By $E[\ell(n)]$ we denote the expected value of $\ell(n)$, and by $V[\ell(n)]$ its variance. Let $ML(n)$ denotes the mean length of the shortest reset word of the automata with n states generated in our experiment.

We have observed that the approximation $ML(n) \approx 1.95n^{0.55}$ proposed in [17] is inflated. We have been searching for an approximation function by filling some predefined templates with different constants and comparing them by minimizing the sum of squares of differences with the experimentally computed estimation. Based on currently available data, we propose a new more precise experimental approximation for the expected length

$$E[\ell(n)] \approx 2.5\sqrt{n-5}. \quad (1)$$

Comparison of both the proposed functions with the experimental results is presented in the Figure 2. We observe that the expected length seems to belong to $\Theta(\sqrt{n})$.

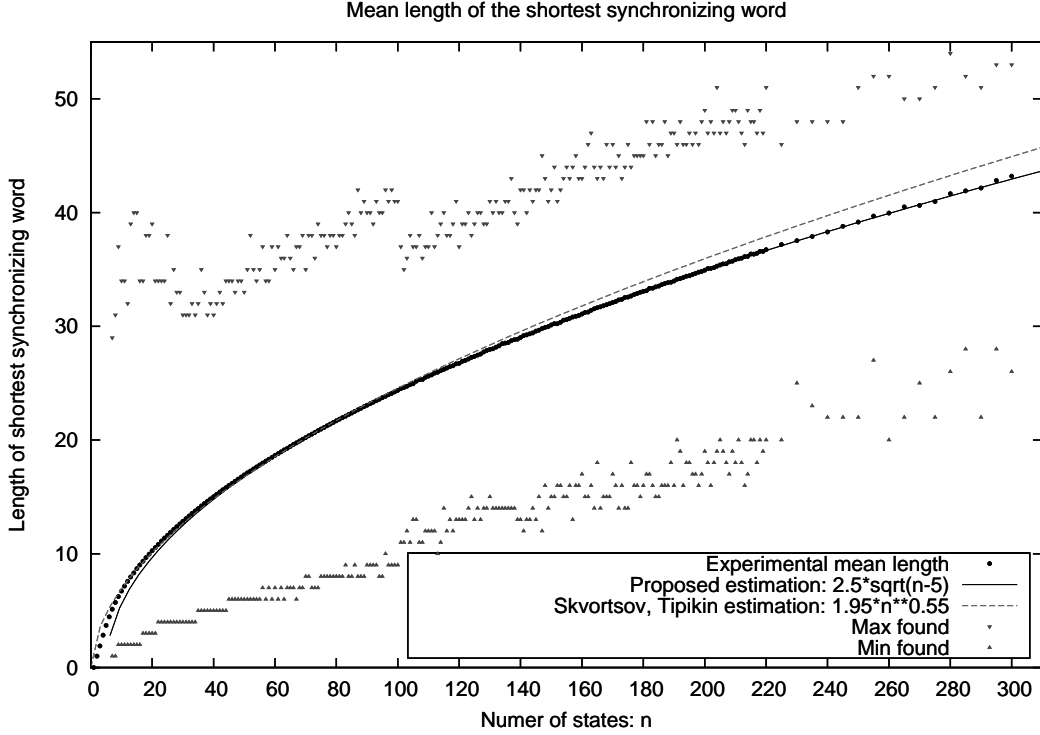


Fig. 2. The approximations proposed for the expected length of the shortest reset word.

3.5 Error estimation

In contrast with the experiments by Skvortsov and Tipikin [17], our experiments allow to obtain a good estimation of the approximation error. We make use of the well-known Hoeffding's inequality.

Yet, since the distribution of $\ell(n)$ is highly asymmetric, one needs to combine this inequality with the statistical fact that the maximal lengths of the shortest reset words are much smaller than the known bounds and that longer lengths occur rarely.

Suitable calculations lead to the following:

Theorem 2 *If less than $1/k$ of the n -state synchronizing automata have the length of the shortest reset word larger than M_n , then with probability $1 - p$*

$$|ML(n) - E[\ell(n)]| \leq \frac{M_n(k-1)}{k} \sqrt{\frac{\log(2/p)}{2m}} + \frac{n^3}{6k}.$$

Assuming the Černý conjecture in the last term $n^3/6$ may be replaced by $(n-1)^2$ (giving essentially better estimation).

For $n = 100$, $m = 10^6$ and $p = 0.0001$ one may show that $k \geq 100975$ is as required, and consequently, the error is less than 1.75 (or 0.19 assuming the Černý conjecture). This means that with high probability the expected length of the shortest reset word for synchronizing automata with $n = 100$ states is close to our experimental result 24.34. Comparing this with the results of Skvortsov and Tipikin [17], we note that, for automata with 100 states, they also have obtained

the expected length close to 24, but taking into account the size of the sample $m = 200$, no reasonable estimation of the error can be obtained in this way (even values of p as large as $p = 0.1$ lead to a few hundred percent error).

3.6 Distribution and variance.

The results of our experiment allow to compute an approximated probability distribution of $\ell(n)$ for each tested n . Example distributions are shown in Figure 3. They are very similar for larger n . For $n = 7$ states the exact distribution is presented.

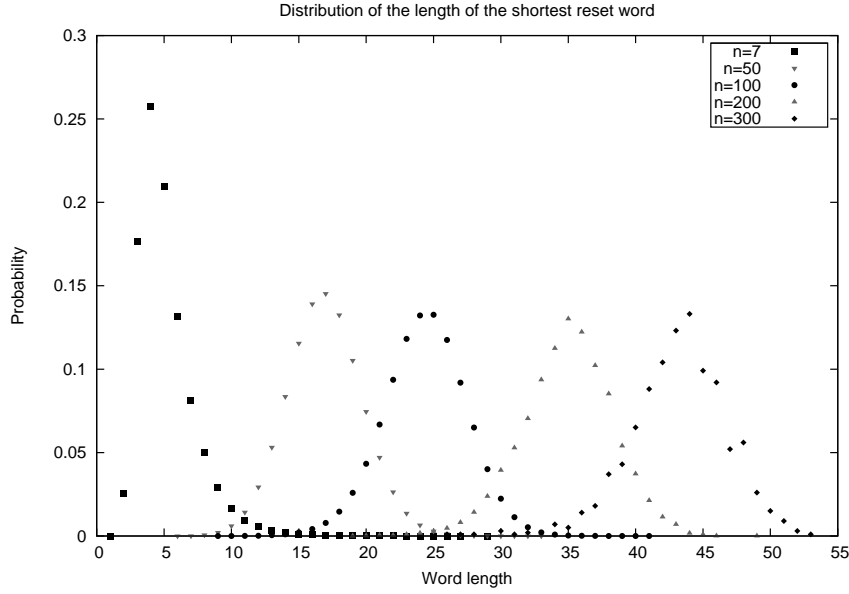


Fig. 3. Probability distributions.

We also confirmed the observations from [17] that the variance $V[\ell(n)]$ is a growing function. We however do not confirm that the fraction $\frac{\sqrt{V[\ell(n)]}}{E[\ell(n)]}$ seems to tend to 0 as n goes to infinity. The graph we have obtained (not reproduced here) does not exclude the possibility that the fraction converges to some positive constant.

4 Conclusion

Our algorithm for finding the minimal reset word is significantly faster than the algorithms used so far and works well for all the automata we have tested. Its time performance depends mainly on the length of the minimum word, but one has to observe that the known slowly synchronizing automata with the longest reset words are not the hardest ones for the algorithm. Thus, there is a hope to discover new examples of classes of slowly synchronizing automata and to check how the situation changes in case of automata with more than 2 input letters. The results of new experiments are intended to be reported in the journal version of the paper. We note that there are still possibilities to optimize the algorithm, in particular, by designing a faster data structure for subset checking and minimalizing subset lists.

References

1. Ananichev, D., Gusev, V., Volkov, M.: Slowly synchronizing automata and digraphs. In: Proceedings of the 35th international conference on Mathematical foundations of computer science. pp. 55–65. MFCS’10, Springer-Verlag (2010)
2. Berlinkov, M.: Approximating the minimum length of synchronizing words is hard. In: Computer Science – Theory and Applications, LNCS, vol. 6072, pp. 37–47. Springer Berlin / Heidelberg (2010)
3. Chmiel, K., Roman, A.: COMPAS - A Computing Package for Synchronization. In: Implementation and Application of Automata, LNCS, vol. 6482, pp. 79–86. Springer Berlin / Heidelberg (2011)
4. Devroye, L.: A note on the average depth of tries. *Computing* 28, 367–371 (1982)
5. Eppstein, D.: Reset sequences for monotonic automata. *SIAM Journal on Computing* 19, 500–510 (1990)
6. Gerbush, M., Heeringa, B.: Approximating minimum reset sequences. In: Proceedings of the 15th international conference on Implementation and application of automata. pp. 154–162. CIAA’10, Springer-Verlag, Berlin, Heidelberg (2011)
7. Higgins, P.: The range order of a product of i transformations from a finite full transformation semigroup. *Semigroup Forum* 37, 31–36 (1988)
8. Kudłacik, R., Roman, A., Wagner, H.: Effective Synchronizing Algorithms, new exact and greedy algorithms to find synchronizing sequences.
9. Martyugin, P.: Complexity of problems concerning reset words for some partial cases of automata. *Acta Cybern.* 19, 517–536 (2009)
10. Martyugin, P.: Complexity of Problems Concerning Reset Words for Cyclic and Eulerian Automata. In: Implementation and Application of Automata, LNCS, vol. 6807, pp. 238–249. Springer Berlin / Heidelberg (2011)
11. Morrison, D.R.: PATRICIA – Practical Algorithm To Retrieve Information Coded in Alphanumeric. *J. ACM* 15, 514–534 (1968)
12. Olschewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In: Proceedings of the 35th international conference on Mathematical foundations of computer science. pp. 568–579. MFCS’10, Springer-Verlag, Berlin, Heidelberg (2010)
13. Panneton, F., L’Ecuyer, P., Matsumoto, M.: Improved long-period generators based on linear recurrences modulo 2. *ACM Trans. Math. Softw.* 32, 1–16 (2006)
14. Roman, A.: New Algorithms for Finding Short Reset Sequences in Synchronizing Automata. In: IEC (Prague). pp. 13–17 (2005)
15. Roman, A.: Genetic Algorithm for Synchronization. In: Language and Automata Theory and Applications, LNCS, vol. 5457, pp. 684–695. Springer Berlin / Heidelberg (2009)
16. Roman, A.: Synchronizing finite automata with short reset words. In: Applied Mathematics and Computation. ICCMSE-2005, vol. 209, pp. 125–136 (2009)
17. Skvortsov, E., Tipikin, E.: Experimental study of the shortest reset word of random automata. In: Proceedings of the 16th international conference on Implementation and application of automata. pp. 290–298. CIAA’11, Springer-Verlag (2011)
18. Skvortsov, E., Zaks, Y.: Synchronizing random automata. *DMTCS* 12(4) (2010)
19. Trahtman, A.: Modifying the Upper Bound on the Length of Minimal Synchronizing Word. In: Fundamentals of Computation Theory, LNCS, vol. 6914, pp. 173–180. Springer Berlin / Heidelberg (2011)
20. Trahtman, A.N.: A package TESTAS for checking some kinds of testability. In: Proceedings of the 7th international conference on Implementation and application of automata. pp. 228–232. CIAA’02, Springer-Verlag, Berlin, Heidelberg (2003)
21. Trahtman, A.: An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In: Mathematical Foundations of Computer Science, LNCS, vol. 4162, pp. 789–800. Springer Berlin / Heidelberg (2006)
22. Volkov, M.: Synchronizing Automata and the Černý Conjecture. In: Language and Automata Theory and Applications, LNCS, vol. 5196, pp. 11–27. Springer Berlin / Heidelberg (2008)